Attention Model using SNN
UCF REU 2022 Week 6

Advisor: Dr. Gonzalo Vaca-Castano
Student: Connor Cabrera
Weekly Overview

- Completed Spiking YOLO v3 Training
  - Finished yesterday, will continue passing though, but want to prioritize development
- Reviewed Vision Transformer Concept
- Wrote code for rate based encoding

Difficulties

- Memory Limitations in Rate Based Encoding
Vision Transformer Progress

Keras

- Ran Keras implementation of ViT model and saved “h5” file
- All layers are compatible with SNN Tool Box, except of course layers.MultiHeadAttention
- Will debug running the h5 through SNNTB to hopefully find an equivalent conversion method

Pytorch

- Haven’t directly compared to Keras
  - SNNTB converts to Keras anyway
- Investigating using a Pytorch ViT and converting all activation functions to Heaviside Step Function
  - Supported by Pytorch only
  - Negative values output 0, Positive values output 1
  - Can approximate derivative
Recall:

- Rate Encoding - Higher intensities have a higher frequency of spikes
  - Due to the inherent randomness of encoding these spikes, multiple spike trains of length “d” (distance) are needed to accurately represent the original image
  - Default values taken from SNN Tool Box
    - d = 200
    - t = 10 (considered not enough)
My Idea

A Pixel Can be Represented by a single 255 length spike train over a single time step

- Spikes are still randomly added to the train based on pixel intensity
- Because the train has a length of 255, the number of spikes is equal to or very near the actual intensity value

Steps:

- Create Output array of Zeros
  - Size = (n, t, height, width, d)
- Probability = Pixel Value / 255
- Generate Random Value between 0 & 1
- If (prob >= rand), spike
Results (MNIST)

Original:

Example Original Pixel:
127

Example Spike Train:
[1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0
  1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0
  0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 0
  1 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1
  1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
  1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0
  1 1 0 0 1 1 0 0 1 0 1 1 0 0 0 1
  0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0
  0 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1
  0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 1
  0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1
  0 0 1 1 0 0 1 0 0 0 1 1 0 1 1 0
  1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1
  0 0 1 0 1 1 0 1 0 1 1 1 1]
Results cont’d (CIFAR10)
Memory Limitations

- By expanding each pixel into a spike train we greatly increase the memory required.
- Was forced to convert only half of CIFAR10
  - Due to color images
  - To the entirety of CIFAR10, I’d need 36.5 GB RAM
  - Will convert entire dataset on Newton
- CIFAR10 is only 32 x 32, memory requirements for larger images unimaginable

HOWEVER

My implementation still takes less memory than the current standard distance x timesteps:

\[(255 \times 1) < (200 \times 10)\]
Thoughts on This

Q: Why couldn’t I find conversion code in SNNTB?
A: I believe since SNNTB simulates an SNN (due to not having neuromorphic hardware) and the concept of conversion revolves around statistical equivalence, SNNTB doesn’t do conversion at all and somehow this equivalence is able to achieve the same goals as an SNN

Q: Why not have the spike trains just be 8 bit representations of pixel values?
Ex: 127 = 01111111
A: Could greatly reduce the memory requirement from being 255 in length, but losing the randomness may provide some disadvantages
What’s Next For Me?

- As mentioned previously, run the keras h5 through SNNTB and debug along the way (safe option)
- Run Pytorch ViT using Heaviside Step Function to try and produce spike train like I/O (exploratory option)
- Run my spike train encoding through regular ANNs, converted ANNs, and the above to see if it holds any merit (stretch option)
- Do the same as above with 8 bit encoding